Even More Dynamic Programming

CSE 421 Fall 22
Lecture 13

Edit Distance

More formally:

The edit distance between two strings is:

The minimum number of **deletions, insertions,** and **substitutions** to transform string x into string y.

Deletion: removing one character

Insertion: inserting one character (at any point in the string)

Substitution: replacing one character with one other.

Example

What’s the distance between `babyyodas` and `tastysoda`?

B	A	B		Y	Y	O	D	A	S
sub		sub	ins		sub				del
T	A	S	T	Y	S	O	D	A	

Distance: 5, one point for each colored box

Quick Checks – can you explain these?
If x has length n and y has length m, the edit distance is at most $\max(x, y)$

The distance from x to y is the same as from y to x (i.e. transforming x to y and y to x are the same)

Finding a recurrence

What information would let us simplify the problem?

What would let us "take one step" toward the solution?

"Handling" one character of x or y

i.e. choosing one of insert, delete, or substitution and increasing the "distance" by 1

OR realizing the characters are the same and matching for free.

$OPT(i, j)$ is the minimum number of insertions, deletions, and substitutions to transform $x_1 x_2 \cdots x_i$ into $y_1 y_2 \cdots y_j$. (we're indexing strings from 1, it'll make things a little prettier).

The recurrence

“Handling” one character of x or y

i.e. choosing one of insert, delete, or substitution and increasing the “distance” by 1

OR realizing the characters are the same and matching for free.

What does delete look like? $OPT(i-1,j)$ (delete character from x match the rest)

Insert $OPT(i,j-1)$ Substitution: $OPT(i-1,j-1)$

Matching charcters? Also $OPT(i-1,j-1)$ but only if $x_i = y_j$

The recurrence (v1, we'll improve soon)

"Handling" one character of x or y

i.e. choosing one of insert, delete, or substitution and increasing the "distance" by 1

OR realizing the characters are the same and matching for free.

$$OPT(i,j) = \begin{cases} \min\{\overset{\boxed{\text{Delete}}}{1+OPT(i-1,j)}, \overset{\boxed{\text{Insert}}}{1+OPT(i,j-1)}, \overset{\boxed{\text{Substitution}}}{1+OPT(i-1,j-1)}, \boxed{\text{TODO: Just Match}}\} & \\ j & \text{if } i = 0 \\ i & \text{if } j = 0 \end{cases}$$

The recurrence (v1, we’ll improve soon)

“Handling” one character of x or y

i.e. choosing one of insert, delete, or substitution and increasing the “distance” by 1

OR realizing the characters are the same and matching for free.

$$OPT(i,j) = \begin{cases} \min\{\overset{\boxed{\text{Delete}}}{1 + OPT(i-1,j)}, \overset{\boxed{\text{Insert}}}{1 + OPT(i,j-1)},\ \overset{\boxed{\text{Substitution}}}{1 + OPT(i-1,j-1)}, \overset{\boxed{\text{Just Match}}}{OPT(i-1,j-1) + \infty \cdot \mathbb{I}\{x_i \neq y_j\}}\} & \\ j & \text{if } i = 0 \\ i & \text{if } j = 0 \end{cases}$$

Idea: only allow “just match” when you can just match.
Otherwise make it ∞ (will never be the min).
In code: if/else branch, probably. This is a math notation trick.

“Indicator” like from 312

The recurrence

“Handling” one character of x or y

i.e. choosing one of insert, delete, or substitution and increasing the “distance” by 1

OR realizing the characters are the same and matching for free.

$$OPT(i,j) = \begin{cases} \min\{\, \overset{\boxed{\text{Delete}}}{1 + OPT(i-1,j)}, \overset{\boxed{\text{Insert}}}{1 + OPT(i,j-1)},\ \overset{\boxed{\text{Sub and matching}}}{\mathbb{I}[x_i \neq y_j] + OPT(i-1,j-1)}\} & \\ j & \text{if } i = 0 \\ i & \text{if } j = 0 \end{cases}$$

When we could match, we will never substitute; matching will always give us a better score! Still have to check delete, insert (those could be better).

The recurrence

“Handling” one character of x or y

i.e. choosing one of insert, delete, or substitution and increasing the “distance” by 1

OR realizing the characters are the same and matching for free.

$$OPT(i,j) = \begin{cases} \min\{\overset{\boxed{\text{Delete}}}{1 + OPT(i-1,j)}, \overset{\boxed{\text{Insert}}}{1 + OPT(i,j-1)},\ \overset{\boxed{\text{Sub and matching}}}{\mathbb{I}[x_i \neq y_j] + OPT(i-1,j-1)}\} & \\ j & \text{if } i = 0 \\ i & \text{if } j = 0 \end{cases}$$

Edit Distance

$\mathbf{OPT}(i,j$	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0										
T 1										
A 2										
S 3										
T 4										
Y 5										
S 6										
O 7										
D 8										
A 9										

Edit Distance

$\mathrm{OPT}(i,j$	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0	0	1	2	3	4	5	6	7	8	9
T 1	1									
A 2	2									
S 3	3									
T 4	4									
Y 5	5									
S 6	6									
O 7	7									
D 8	8									
A 9	9									

Edit Distance

B	A	B	Y
T	A	S	

Current subproblem: edit dist between BABY and TAS

$\mathrm{OPT}(i,j$	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0	0	1	2	3	4	5	6	7	8	9
T 1	1									
A 2	2									
S 3	3									
T 4	4									
Y 5	5									
S 6	6									
O 7	7									
D 8	8									
A 9	9									

Edit Distance

B	A	B	Y
T	A	S	

Delete: get rid of Y (cost 1)
BAB
TAS

$\mathrm{OPT}(i,j$	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0	0	1	2	3	4	5	6	7	8	9
T 1	1									
A 2	2									
S 3	3			🟥	🟫					
T 4	4									
Y 5	5									
S 6	6									
O 7	7									
D 8	8									
A 9	9									

Edit Distance

B	A	B	Y
T	A	S	

Insert: insert S (cost 1)

BABY

TA

OPT(i, j	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0	0	1	2	3	4	5	6	7	8	9
T 1	1									
A 2	2				🟪					
S 3	3			🟥	🟫					
T 4	4									
Y 5	5									
S 6	6									
O 7	7									
D 8	8									
A 9	9									

Edit Distance

B	A	B	Y
T	A	S	

Sub: Transform Y to S (cost 1)

BAB

TA

OPT$(i,j$	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0	0	1	2	3	4	5	6	7	8	9
T 1	1									
A 2	2			🟩	🟪					
S 3	3			🟥	🟫					
T 4	4									
Y 5	5									
S 6	6									
O 7	7									
D 8	8									
A 9	9									

Edit Distance

B	A	B	Y
T	A	S	

Gold entry will be min of:
$1 +$ delete
$1 +$ insert
$1 +$ sub

$\mathrm{OPT}(i,j$	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0	0	1	2	3	4	5	6	7	8	9
T 1	1									
A 2	2			🟩	🟪					
S 3	3			🟥	🟨					
T 4	4									
Y 5	5									
S 6	6									
O 7	7									
D 8	8									
A 9	9									

Edit Distance

$1 + 2$
$1 + 3$
$1 + 2$
Min: 3

Gold entry will be min of:
$1 +$ delete
$1 +$ insert
$1 +$ sub

$\text{OPT}(i, j$	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0	0	1	2	3	4	5	6	7	8	9
T 1	1	1	2	3	4	5	6	7	8	9
A 2	2	2	1	2	3	4	5	6	7	8
S 3	3	3	2	2	3	4	5	6	7	7
T 4	4	4	3	3	3	4	5	6	7	8
Y 5	5	5	4	4	3					
S 6										
O 7										
D 8										
A 9										

Edit Distance

Fill in the next two entries. Be careful with the sub/match distinction!

$\mathbf{OPT}(i, j$	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0	0	1	2	3	4	5	6	7	8	9
T 1	1	1	2	3	4	5	6	7	8	9
A 2	2	2	1	2	3	4	5	6	7	8
S 3	3	3	2	2	3	4	5	6	7	7
T 4	4	4	3	3	3	4	5	6	7	8
Y 5	5	5	4	4	3					
S 6										
O 7										
D 8										
A 9										

Edit Distance

Fill in the next two entries. Be careful with the sub/match distinction!

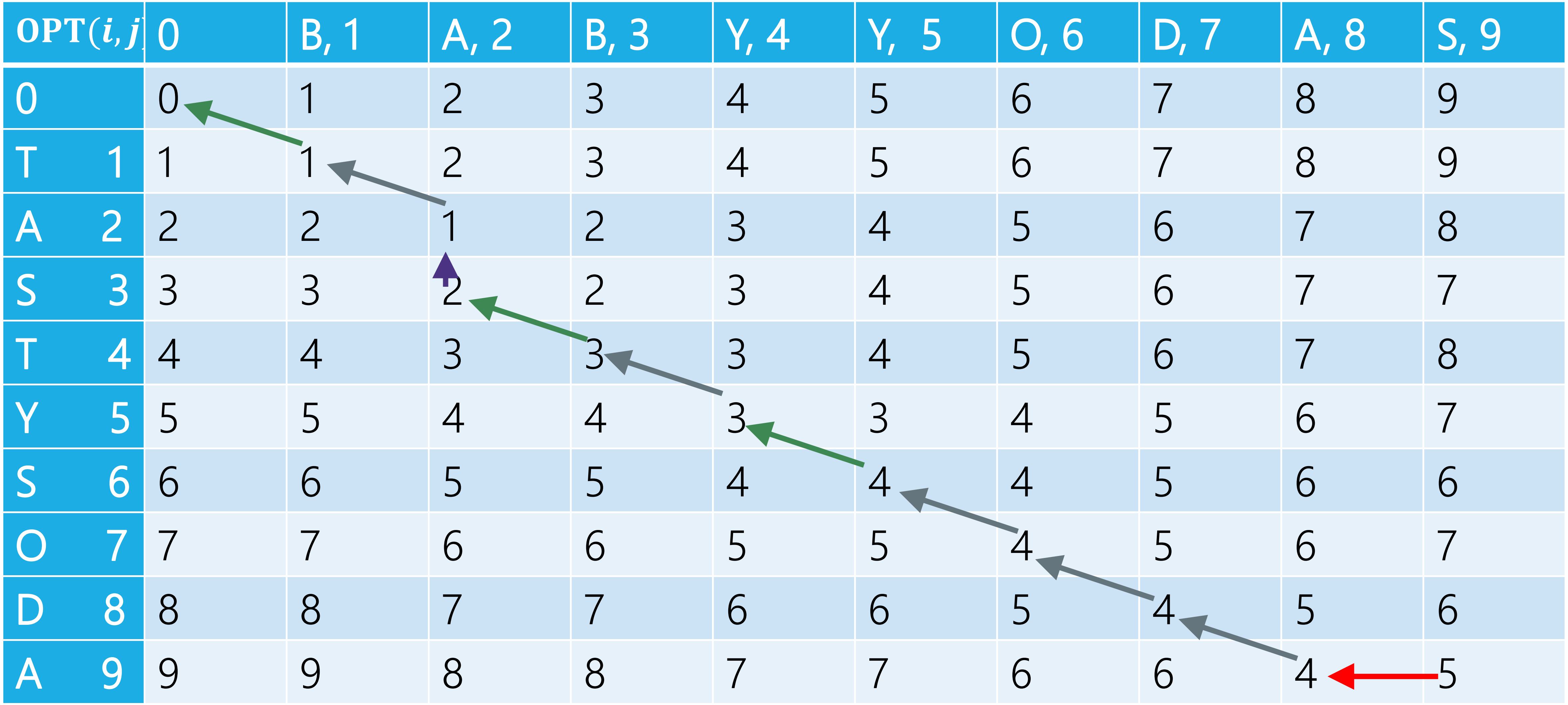
$\mathrm{OPT}(i,j)$	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0	0	1	2	3	4	5	6	7	8	9
T 1	1	1	2	3	4	5	6	7	8	9
A 2	2	2	1	2	3	4	5	6	7	8
S 3	3	3	2	2	3	4	5	6	7	7
T 4	4	4	3	3	3	4	5	6	7	8
Y 5	5	5	4	4	3	**3**	**4**			
S 6										
O 7										
D 8										
A 9										

Y's match, so sub is free!

Edit Distance

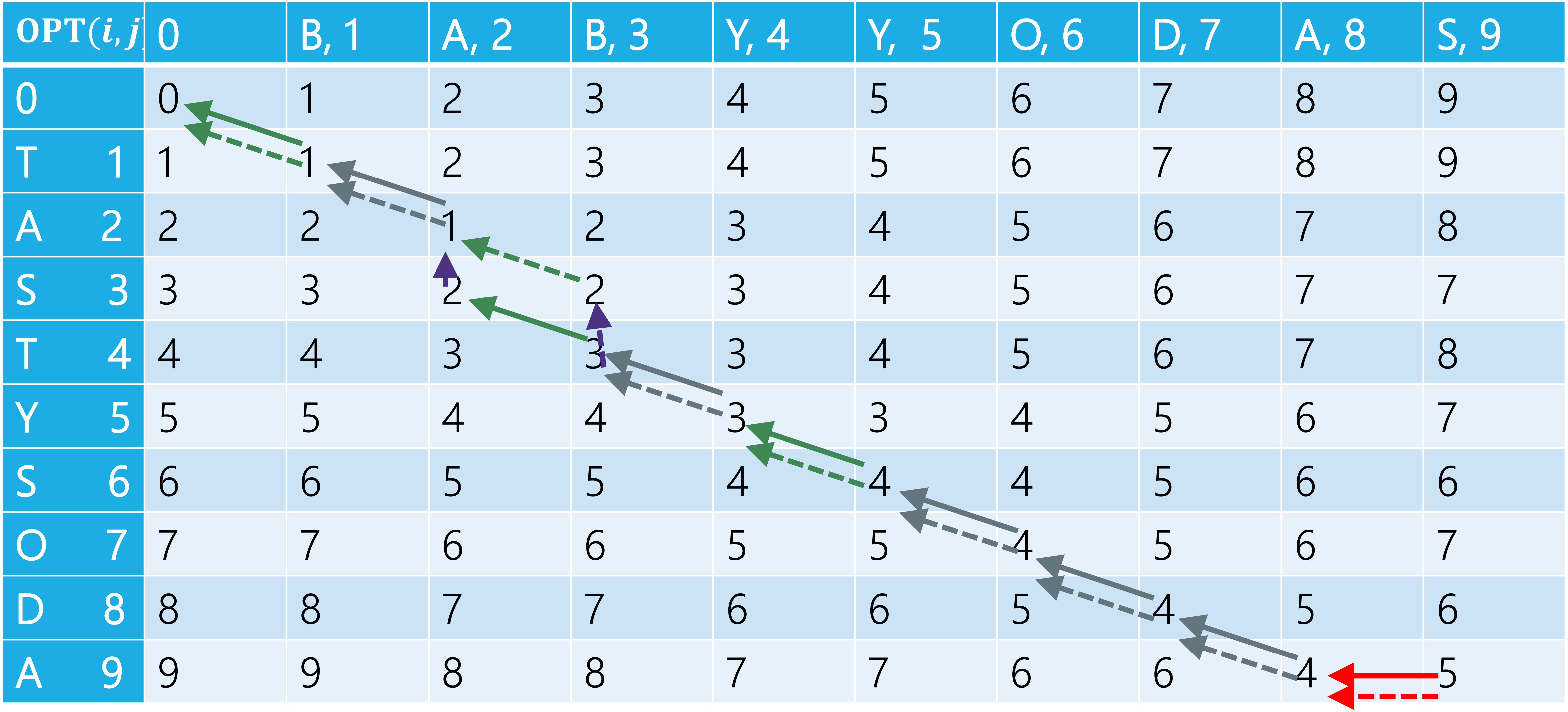
OPT$(i,j$	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0	0	1	2	3	4	5	6	7	8	9
T 1	1	1	2	3	4	5	6	7	8	9
A 2	2	2	1	2	3	4	5	6	7	8
S 3	3	3	2	2	3	4	5	6	7	7
T 4	4	4	3	3	3	4	5	6	7	8
Y 5	5	5	4	4	3	3	4	5	6	7
S 6	6	6	5	5	4	4	4	5	6	6
O 7	7	7	6	6	5	5	4	5	6	7
D 8	8	8	7	7	6	6	5	4	5	6
A 9	9	9	8	8	7	7	6	6	4	5

Edit Distance – what operations?



$\mathrm{OPT}(i,j$	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0	0	1	2	3	4	5	6	7	8	9
T 1	1	1	2	3	4	5	6	7	8	9
A 2	2	2	1	2	3	4	5	6	7	8
S 3	3	3	2	2	3	4	5	6	7	7
T 4	4	4	3	3	3	4	5	6	7	8
Y 5	5	5	4	4	3	3	4	5	6	7
S 6	6	6	5	5	4	4	4	5	6	6
O 7	7	7	6	6	5	5	4	5	6	7
D 8	8	8	7	7	6	6	5	4	5	6
A 9	9	9	8	8	7	7	6	6	4	5

Edit Distance – what operations?



OPT$(i,j$	0	B, 1	A, 2	B, 3	Y, 4	Y, 5	O, 6	D, 7	A, 8	S, 9
0	0	1	2	3	4	5	6	7	8	9
T 1	1	1	2	3	4	5	6	7	8	9
A 2	2	2	1	2	3	4	5	6	7	8
S 3	3	3	2	2	3	4	5	6	7	7
T 4	4	4	3	3	3	4	5	6	7	8
Y 5	5	5	4	4	3	3	4	5	6	7
S 6	6	6	5	5	4	4	4	5	6	6
O 7	7	7	6	6	5	5	4	5	6	7
D 8	8	8	7	7	6	6	5	4	5	6
A 9	9	9	8	8	7	7	6	6	4	5

Dynamic Programming Process

1. Define the object you're looking for

   $OPT(i, j)$ is the minimum number of insertions, deletions, and substitutions to transform $x_1 x_2 \cdots x_i$ into $y_1 y_2 \cdots y_j$.

2. Write a recurrence to say how to find it

   ✅

3. Design a memoization structure

   $m \times n$ Array

4. Write an iterative algorithm

   Outer loop: increasing j (starting from 1)
   Inner loop: increasing i (starting from 1)

DP Proofs

We generally **won’t** ask you for proofs of correctness on dynamic programming problems.

Why?

The proofs are always inductive proofs where you say

“my recurrence checks all the possibilities” or, equivalently

“The maximum thing has to be made up of the best thing for all these other subproblems.“

The proof itself is very difficult to write clearly (you have to differentiate between your recurrence and what your recurrence intends to calculate, which can be

DP Proofs

We’ll include an example proof sometime in the next few week so you know what you’re (not) missing.

Instead, we’re going to ask for your intuition on what your recurrence is doing (what do all the cases correspond to/why are they exhaustive)?

The proof is just a lot of formalism on that key idea. So we’re going to have you focus on the idea, not the formalism.

Goal of DP

Just try all the (reasonable) possibilities.

Don't worry about greedily choosing the best, use recursion to "look ahead" for all the best options, and pick the best one.

There is a "greedy-ish" alteration to the Edit Distance recurrence…

It turns out, if the two characters match, that will always be at least as good as the insert/delete options.

But it's fine to **not** notice! And if you thought it was safe but wasn't, well….

More Problems

Maximum Contiguous Subarray Sum

We saw an $O(n \log n)$ divide and conquer algorithm.

Can we do better with DP?

Given: Array $A[]$

Output: i, j such that $A[i] + A[i+1] + \cdots + A[j]$ is maximized.

Dynamic Programming Process

1. Define the object you’re looking for

2. Write a recurrence to say how to find it

3. Design a memoization structure

4. Write an iterative algorithm

Maximum Contiguous Subarray Sum

We saw an $O(n \log n)$ divide and conquer algorithm.

Can we do better with DP?

Given: Array $A[]$

Output: i, j such that $A[i] + A[i+1] + \cdots + A[j]$ is maximized.

For today: just output the value$A[i] + A[i+1] + \cdots + A[j]$.

Is it enough to know `OPT(i)`?

Trying to Recurse

0	1	2	3	4	5	6	7
5	-6	3	4	-5	2	2	4

$OPT(3)$ would give i =2, $j = 3$

$OPT(4)$ would give $i = 2, j = 3$ too

$OPT(7)$ would give $i = 2, j = 7$ – we need to suddenly backfill with a bunch of elements that weren't optimal…

How do we make a decision on index 7? What information do we need?

What do we need for recursion?

If index i IS going to be included

We need the best subarray **that includes index $\boldsymbol{i-1}$**

If we include anything to the left, we'll definitely include index $i-1$ (because of the contiguous requirement)

If index i isn't included

We need the best subarray up to $i-1$, regardless of whether $i-1$ is included.

Two Values

Pollev.com/robbie

Need two recursive values:

$INCLUDE(i)$: sum of the maximum sum subarray among elements from 0 to i **that includes index $\boldsymbol{i}$** in the sum

$OPT(i)$: sum of the maximum sum subarray among elements 0 to i (that might or might not include i)

How can you calculate these values? Try to write recurrence(s), then think about memoization and running time.

Recurrences

$$INCLUDE(i) = \begin{cases} \max\{A[i], A[i] + INLCUDE(i-1)\} & \text{if } i \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

$$OPT(i) = \begin{cases} \max\{INCLUDE(i), OPT(i-1)\} & \text{if } i \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

If we include i, the subarray must be either just i or also include $i-1$.

Overall, we might or might not include i. If we don't include i, we only have access to elements $i-1$ and before. If we do, we want $INCLUDE(i)$ by definition.

Example

	0	1	2	3	4	5	6	7
A	5	-6	3	4	-5	2	2	4

	0	1	2	3	4	5	6	7
$OPT(i)$	5							

	0	1	2	3	4	5	6	7
$INCLUDE(i)$	5							

Example

	0	1	2	3	4	5	6	7
A	5	-6	3	4	-5	2	2	4

	0	1	2	3	4	5	6	7
$OPT(i)$	5	5						

	0	1	2	3	4	5	6	7
$INCLUDE(i)$	5	-1						

Example

	0	1	2	3	4	5	6	7
A	5	-6	3	4	-5	2	2	4

	0	1	2	3	4	5	6	7
$OPT(i)$	5	5	5	7	7	7	7	10

	0	1	2	3	4	5	6	7
$INCLUDE(i)$	5	-1	3	7	2	4	6	10

Pseudocode

```
int maxSubarraySum(int[] A)
  int n=A.length
  int[] OPT = new int[n]
  int[] Inc = new int[n]
  inc[0]=A[0]; OPT[0] = max{A[0],0}
  for(int i=0;i<n;i++)
    inc[i]=max{A[i], A[i]+inc[i-1]}
    OPT[i]=max{inc[i], opt[i-1]}
  endFor
return OPT[n-1]
```

Recursive Thinking In General

As before, the hardest part is designing the recurrence.

It sometimes helps to think from multiple different angles.

Top-down: What's the first step to take?

Baby Yoda will first go left or down. Use recursion to find out which of left or down is better.

The farthest right operation in the string transformation will be one of insert, delete, substitute, match for free. Use recursion to find out which is best.

Recursive Thinking In General

Bottom-Up: What information could a recursive call give me that would help?

How does a path through most of the map help Baby Yoda?

Well we just need to know the values one left and one down.

The edit distance between which strings would help us compute the edit distance between our strings?

Well if we know the distance between $x_1 \ldots x_{i-1}$ and $y_1 \ldots y_{j-1}$ then that would tell us what happens if we substitute…that might lead you to insertions and deletions too.

Recursive Thinking In General

Some people refer to the “Optimal Substructure Property”

From the optimum (most eggs, fewest number of string operations) for a slightly smaller problem (Baby Yoda starting closer to the end, slightly smaller strings), we need to be able to build up the optimum for the full problem.

Longest Increasing Subsequence

0	1	2	3	4	5	6	7
5	-6	3	6	-5	2	8	10

Longest set of (not necessarily consecutive) elements that are increasing

5 is optimal for the array above

(indices $1,2,3,6,7$; elements $-6,3,6,8,10$)

For simplicity – assume all array elements are distinct.